Investigation into the Interplay Between Encryption and Compression

Introduction

As the volume of data sent over the Internet increases, the value of data compression becomes apparent. By decreasing the total number of bits sent, compression allows for a decrease in cost to send and a decrease in time spent sending information.

However, security is also an issue. When you send a packet, it does not simply go from your location to its destination. It may go through any number of secure or insecure machines before it reaches its final destination. Each exchange point is an opportunity for an information leak.

For governments, large corporations, and individuals, this represents a serious risk. To combat that risk, encryption is typically used on certain messages. Remember, encryption does not make it impossible for an unintended recipient to read an intercepted message; rather it makes it prohibitively difficult to understand that message.

But there is a problem: encryption and compression are seemingly opposed to each other on a fundamental level. Encryption aims to obfuscate the true nature of a message, making it resemble a random string more. Compression takes advantage of patterns and reduces the file based on those patterns; making it resemble a random string less.

Traditionally, this problem is handled by compressing the sensitive data before encrypting it (Johnson, p1). However, as Johnson et al. discuss, by using a stream cipher, one may be able to reverse that order; relieving the burden of major computation from the client. A stream cipher is a type of encryption strategy that jumbles small, discrete parts of the message stream; typically with a number of
applications of XOR. Although this may not be useful to desktop machines or machines with significant power, it could be used by small, mobile devices. This strategy would allow for small, mobile devices and other embedded systems to have access to some degree of encryption without incurring the cost of compression, because the servers between the source and destination will handle compression and decompression.

Another work of note along the same lines was investigated by Gurijala and Khayam. They investigated the correlation between peak signal-to-noise ratio (or PSNR) and compression rate as a function of bits per pixel (Gurijala, p1). They concluded from their results that "At low compression rates, encryption can be employed without significant compromise in compression efficiency. Such low bit-rate compression has applications in bandwidth-constrained wireless environments." These results seem to back up the claims made by Johnson et al that a limited form of encryption could be used to offer some security to messages without being too costly.

In my work, I aimed to investigate the interplay between compression and encryption. I planned to measure the compression rate as well as the difference in first-order entropy in terms of the encryption's difficulty. The goal was to see if anything meaningful could be gleaned from using an actual application of the methods discussed which could not be seen from theory and conjecture alone.

Procedure

The first thing I needed was to get a data set. I chose the Yale Face Database3. This contains 164 images of various people in differing lighting conditions. These images all came in GIF format. To make the encryption and compression process more observable, I converted all of the images to PGM4 (Portable Gray Map) format. This is a basically raw image format, very similar to IMG, where each pixel is represented by a number from 0 to 255. I was able to do this with a Linux utility called giftopnm.
After I had my faces, I needed to come up with a few compression schemes to investigate. I chose the Huffman and Adaptive Huffman provided in the code accompanying the class's book, "Introduction to Data Compression". To measure the first-order entropy of the streams, I used my code from the first assignment.

For encryption, I created three programs of different type. I created a stream cipher which uses multiple applications of a bitwise XOR to encrypt the stream. By applying more XORs with different keys, I was able to dictate the difficulty of the cipher stream. I also created a program which encrypts the stream using the Blowfish algorithm. The Blowfish algorithm allows for a variable length key, which I used to determine the "difficulty" of the encryption. My third encryption scheme was an autokey cipher. This used simple offset and modular arithmetic to hide the true nature of the data sent in.

The next step was to provide a mathematical and reproducible means of verifying whether the encrypted streams were safe. I chose to use a face recognition program to try and see a face in any given stream. If the stream was encrypted and a face could not be seen, it was determined to be safe. My implementation used the OpenCV library.

I based my program off of a simple sample program that they provide. It is given a file. If the file is an image, it will try to detect a face. If a face is seen, it will print 1. Otherwise, it will print 0. If the file is not an image, it is interpreted as one given its raw byte format (because the PGM format is essentially a raw byte format). It will then take the interpreted stream and try to find a face in that.

Finally, I had to construct a means to perform all of my tests. I had 164 images. On each image, I would have to perform a check of disk usage, first-order entropy, and face recognition. These three operations would have to be done on three occasions: before the image is altered, after compression and encryption, and after it is restored. If the process is not reversible, then we cannot get back our original message and the whole thing is worthless.
We must perform the operations described above for two different strategies: one where compression precedes encryption (traditional or standard order), and the other where encryption comes first (novel order). There are three encryption methods and two compression methods. Finally, everything described above must be tested on 8 different difficulty levels. The grand total of data points collected then was $164 \times 3 \times 3 \times 3 \times 2 \times 2 \times 8 = 141696!$

Because of the sheer volume of data I would be collecting, I created a series of Bash shell scripts which would handle the operations. They printed all of the results into CSV (Comma Separated Value) files in the `res` directory.

From there, I used a spreadsheet application to take the averages of the results. The 164 entries in each CSV file averaged to give me two data points per file: one for compression rate versus encryption, and one for first-order entropy versus encryption. I then graphed my results to see if any correlation could be found.

**Results**

For my results, I will first describe the format of my graphs so that the data may be understood easily. Each graph has a title which shows the compression and encryption methods used. There are two types of graphs: compression rate versus encryption and first-order entropy versus encryption. Along the x-axis is the difficulty of the encryption at 8 discrete points, where 1 is the easiest and 8 is the hardest.

For the compression graphs, the y-axis is final size. That means that the smaller the number is, the greater compression was achieved. A value of 1 means no compression was achieved at all. For first-order entropy, the y-axis represents a ratio of gain in entropy. A higher number means a greater increase in entropy. A value of 1 would mean no increase in entropy. A value of 2 would mean a doubling of entropy. Most measurements for this variable were above the 1.6 mark.
The order is also important. *Order 1* is the novel method discussed by Johnson et al. It represents encrypting a stream before compression. *Order 2* is the standard method of applying compression before encryption.

Below I will present my results for the experiment. I will discuss my results in the conclusions section.
I will now discuss each graph of the data I collected. The first will be the graphs with the data collected from the configuration using Blowfish encryption and Huffman compression. This data shows that the results seem to be independent of the encryption difficulty. This may be a function of the Blowfish algorithm. It is possible that Blowfish provides a steady amount of compression regardless of the key length. Another possibility is that the input streams were too small to be effectively encrypted by Blowfish.

Whatever the reason, as we can see, the traditional method prevails here. There was no difference in the first-order entropy rates (so the data with Order 1 was no more safe), but the compression on the traditional order clearly outclassed the novel order.

Next is my XOR cipher code with Huffman compression. Order 1's compression rate was worse than Order 2's all around except for when the difficulty was 8, where it matched. In terms of first-order entropy, Order 1 performed worse all-around again, except for when the difficulty was 8, where it matched Order 2 again. The entropy gains of both orders seemed to be greatly affected by the number of applications of XOR, where an odd number of applications produces a higher gain in entropy than an even number of applications.
Next we have the Blowfish encryption along with Adaptive Huffman compression. As you can see, there is no variation at all in these numbers. The traditional order performs the best under these circumstances.

After that, we have the XOR cipher code with the Adaptive Huffman compression. This setup is similar to the setup discussed in the cited papers. Here we have a compression algorithm that is made for streaming data and an encryption algorithm that is also designed for streaming data.

As you can see, the traditional order is still better in terms of compression. However, the gains in entropy are much closer between the two strategies than they were when regular Huffman was used to compress. Not only that, but the gains in entropy were greater than with the regular Huffman compression.

Next we have the autokey cipher with Huffman compression. Here the traditional order was also superior to the novel order. In both final size and entropy, the novel order performed worse than the traditional order. However, notice the spikes at difficulties 3 and 6.

In the final set of graphs, we examine the autokey cipher with Adaptive Huffman compression. Again, in terms of compression, the novel order performed worse than the traditional order. However, in terms of entropy gain, it performed better than the traditional order with a few curious exceptions.

Those exceptions take place at difficulties 3 and 6 on all graphs involving the autokey cipher encryption. For whatever reason, those settings, which correspond to 3 or 6 applications of the autokey cipher, lend themselves to a better compression rate by a lower gain in entropy.

**Conclusions**

From this data we can see that given a standard implementation, the traditional method of compression before encryption is best. When you have the computing power or time to perform these
computations, you should because they produce smaller and safer results overall.

However, in the case of encryption of streaming data for a small device, the benefits of the novel method can be seen. These benefits are best shown by the set of graphs with the cipher codes and Adaptive Huffman compression. That is because these most closely represent the setup that would be used by the novel strategy.

Although the overall compression suffers to some degree, there are some configurations that allow the compression rates to match (or at least come close to) those of the traditional method. Additionally, those configurations are typically as safe or safer than similar configurations of the traditional method.

For standard set-ups, my data support what intuition and practice tell us about encryption and compression: that compression should be performed before encryption. However, it also suggests that with the proper configuration, the novel approach may also be tenable. A full investigation would need to be performed to decide what configuration performs the best in that situation and what, if any, benefits there are.

It must also be considered that there are more factors than the final size of the message and the safety of the message. Computing power, energy consumption, and time spent computing are just some of the concerns that are related to the potential gains to be made by using the novel order properly.

For instance, limited computing power, energy, and a need to real-time transmission are the reasons why the traditional method is cumbersome for small devices. Obviously, the novel method is completely superfluous for desktop machines.

But it is important to investigate the gains made by using encryption before compression: not just in terms of safety, but in terms of other costs too. If performing the encryption and compression in this order makes the device's CPU burden less, what must we give up? The message may not be as
small, or it may be less safe to transmit. These are the concerns investigators must address in a full experimental investigation of the novel strategy.

Of course, those losses may be acceptable, but they are ones that should be investigated fully nonetheless. As data are streamed from mobile devices in increasing volume, the value of this becomes apparent. Without the need to perform heavy computations, a device which sends small, safe messages using the novel method may be preferable over one that sends small, safe messages using the standard method because its size and cost may be reduced.

In conclusion, the data I collected showed what one might expect them to with some exceptions. These exceptions were theorized in the works of Johnson et al. Although the gains found in my data are exceptional by no means, they do reveal a niche where additional performance gains could be sought. In the coming years, as the demand for both compression and encryption increase, a method that might improve performance and cost should be researched.
Citations


